



TU Clausthal

Clausthal University of Technology

A Novel Approach to Workflow Scheduling in MediGRID

Dietmar Sommerfeld¹ and Harald Richter²

IfI Technical Report Series

IfI-09-07

The logo for the Institute of Information Systems (IfI) at TU Clausthal, consisting of the letters 'IfI' in a stylized, bold, white font.A white diamond shape with a black outline, positioned on the left side of the bottom green bar.

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany

Editor of the series: Jürgen Dix

Technical editor: Michael Köster

Contact: michael.koester@tu-clausthal.de

URL: <http://www.in.tu-clausthal.de/forschung/technical-reports/>

ISSN: 1860-8477

The IfI Review Board

Prof. Dr. Jürgen Dix (Theoretical Computer Science/Computational Intelligence)

Prof. Dr. Klaus Ecker (Applied Computer Science)

Prof. Dr. Barbara Hammer (Theoretical Foundations of Computer Science)

Prof. Dr. Sven Hartmann (Databases and Information Systems)

Prof. Dr. Kai Hormann (Computer Graphics)

Prof. Dr. Gerhard R. Joubert (Practical Computer Science)

apl. Prof. Dr. Günter Kemnitz (Hardware and Robotics)

Prof. Dr. Ingbert Kupka (Theoretical Computer Science)

Prof. Dr. Wilfried Lex (Mathematical Foundations of Computer Science)

Prof. Dr. Jörg Müller (Business Information Technology)

Prof. Dr. Niels Pinkwart (Business Information Technology)

Prof. Dr. Andreas Rausch (Software Systems Engineering)

apl. Prof. Dr. Matthias Reuter (Modeling and Simulation)

Prof. Dr. Harald Richter (Technical Computer Science)

Prof. Dr. Gabriel Zachmann (Computer Graphics)

Prof. Dr. Christian Siemers (Hardware and Robotics)

A Novel Approach to Workflow Scheduling in MediGRID

Dietmar Sommerfeld¹ and Harald Richter²

¹ Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
Am Fassberg 11, 37077 Göttingen, Germany
dsommer@gwdg.de

² Department of Informatics, Clausthal University of Technology
Arnold-Sommerfeld-Str. 1, 38678 Clausthal-Zellerfeld, Germany
hri@tu-clausthal.de

Abstract

We describe four problems inherent to Grid scheduling that could be identified by means of measurements in the D-Grid. These problems make meta-scheduling nearly always a delicate task. In the face of this, we developed a new hybrid methodology to schedule application workflows which presumably supersedes existing methods. Our algorithm combines existing scheduling strategies for the Grid and for workflows, and it additionally employs three prediction methods for the expected queue waiting times. Three site scenarios could be identified where one respective prediction works best. To meet the dynamic characteristics of heterogeneous Grid resources, we use a list scheduling heuristic to perform full-ahead planning of workflow tasks based on execution time predictions, and then distribute Grid jobs just-in-time according to resource performance predictions calculated from up-to-date monitoring data.

1 Introduction

In this paper, we propose a novel approach for the scheduling of Grid workflows within the MediGRID project which is part of the German e-Science initiative D-Grid. MediGRID is a community Grid for researchers in the fields of medicine, biomedical informatics, and life sciences. Four types of pilot applications were selected for the first phase of MediGRID: bioinformatics, image processing, biomedical ontology, and clinical research applications. MediGRID has set up a service Grid that uses the Globus Toolkit 4 (GT4) as its Grid middleware. GT4 is based on stateful web services and provides basic services for security infrastructure, job execution, data management, monitoring and resource discovery.

On top of GT4, MediGRID employs an advanced workflow system for orchestrating the distributed execution of workflow applications on Grid resources. The important characteristic of these applications is that they are not monolithic and single-executable applications, but incorporate multiple dependent computational modules, and entail transfer and storage of a large amount of data. The execution of these applications involves the concurrent and sequential execution of multiple programs, and the automatic and timely data transfer between programs which are also called tasks in workflow terms. A very important issue in executing a scientific workflow application in computational Grids is how to map and schedule workflow tasks onto multiple distributed resources, and how to handle task dependencies in a timely manner to deliver the requested performance.

We describe a methodology to schedule application workflows that combines existing scheduling strategies for workflow tasks and Grid jobs in a performance-effective approach with manageable complexity. This approach consists of two tiers. In the first tier, a full-ahead schedule of the workflow tasks is created. This means that in a first step the whole workflow is scheduled statically before its execution. For this, we use a list scheduling heuristic similar to the Heterogeneous Earliest-Finish-Time algorithm (HEFT) [21]. In the resulting static schedule, all tasks are assigned ranks which determine static prioritizations. The second tier operates dynamically at runtime and processes the workflow tasks according to their priorities. It performs a just-in-time mapping of tasks to Grid resources and is equivalent to common meta-scheduling on the Grid. To improve the just-in-time scheduling decision, it uses dynamic resource information and short-time predictions.

2 State of the Art

Scheduling of scientific workflow applications on the Grid is still a challenge although the number of approaches is as numerous as the projects dealing with this topic. Related work on the problem is mostly focused on either the domains of workflow scheduling or meta-scheduling. A comprehensive overview of meta-scheduling on the Grid is given in [17].

An Important aspect of workflow scheduling is the trade-off between just-in-time and long term look-ahead operation. In [9] Deelman et. al. propose a scheduling strategy based on initial partitioning of the workflow into sequential sub-workflows that are scheduled one after another. Prodan [18] applies genetic algorithms to schedule the whole workflow at once. Then it is rescheduled many times during the execution.

In [23] Wiczeorek compares three scheduling algorithms for scheduling scientific workflows in Grid environments. The scheduling algorithms comprise a genetic algorithm, the HEFT algorithm, and a just-in-time algorithm, similar to the Condor DAGMan resource broker [20], which schedules the

next task onto the best machine available without any long-term optimization strategy. Additionally, different scheduling strategies are compared including full-graph scheduling and an incremental workflow-partitioning strategy. In a series of experiments, it is shown by Wieczorek that the HEFT algorithm applied with the full-ahead scheduling strategy performs best in practice compared to other approaches.

A general analysis of multi-criteria grid workflow scheduling is given in [22]. It proposes novel taxonomies of the problem, based on the aspects (e.g. models, criteria) which influence the decision which scheduling strategy is most appropriate in a given case. Many of the existing workflow scheduling approaches for the Grid are analyzed and classified according to the proposed taxonomies.

Recently, the Gridway [13] meta-scheduler has become a popular solution for job scheduling in Globus based Grids. Gridway is part of the GT4 distribution and relies solely on Globus components. The latest version also incorporates support for Condor DAGMan workflows.

3 Measurements in D-Grid

In the past months, we have conducted a number of experiments to investigate the availability of resources at D-Grid computing sites. A measurement program was installed on five big clusters which are in production use by the MediGRID community. These clusters are located at different D-Grid sites and comprise in total more than 5000 processor cores. All clusters consist of many machines and are controlled by a local resource management system (LRMS) which schedules the jobs in the input queues onto the processors. The measurement program periodically submits test jobs and measures thus queue waiting times. It additionally retrieves the total number of running and waiting jobs. As a result of our measurements, we have disclosed four intrinsic problems for meta-scheduling that have to be considered in the future in the design of Grid meta-schedulers.

These problems are:

1. Resources used in Grid computing typically consist of large clusters that are used by several Grid projects and multiple virtual organizations (VO). Each of these VOs can use its own Grid scheduler. In this case, every Grid scheduler only has the role of a “power user” that competes with other users, e.g. other Grid schedulers in the same grid from the perspective of the LRMS. As a consequence, an optimal schedule is not possible for any individual meta-scheduler. We call this the competing-schedulers problem.
2. Computing sites that take part in Grid projects retain the concept of site autonomy. This means that administrative decisions and privileges

remain at the site level. Therefore, Grid schedulers have no control over the site scheduler's policy, and over the prioritization of the jobs waiting in the queues. Thus, no control exists over the respective LMRS for the meta-scheduler. We call this the lack-of-control problem.

3. Every site uses a custom configuration of queues, and processors can be shared among queues or can be dedicated to queues exclusively. Usually the information from the LRMS does not allow to reliably determine the number of available processors. The only statistics commonly available are the number of running and waiting jobs. Some sites do not even provide this information because of nondisclosure agreements. Additionally, sites employ custom configurations for their local scheduling which are usually non-transparent to the users. To conclude, local schedulers currently do not provide sufficient information for a good schedule at the Grid level. We call this the information-insufficiency problem.
4. Resources are normally highly utilized, and waiting times at clusters can last up to hours. Therefore, input queue waiting time considerably exceeds the actual execution time for small jobs and is their dominant factor. However, input queue waiting time and input queue length have shown to be not continuously differentiable functions over time. Instead, they can vary within minutes by a factor of thousand or more. They behave more like fractals than functions. This makes predictions of future queue waiting times and queue lengths a delicate task. However, scheduling always relies on such predictions. We call this the non-continuously differentiable-function problem.

Figs. 1-5 show the variations of waiting time, running and waiting jobs for all five sites during the same measurement interval which was 720 hours (30 days). The left y-axis denotes the waiting time in minutes, the right y-axis the number of waiting and running jobs. The measurement program that collects the data was executed by means of a regular MediGRID user account. This account was used solely for test jobs and did not run production jobs that would consume a notable amount of resources. From site to site, there are a number of typical scheduling criteria: Jobs can be assigned static priorities based on user, group and job class. Fairshare-scheduling prioritizes jobs based on the historical resource usage. Furthermore, jobs can be prioritized depending on the amount of requested resources. The most common method is to raise priority with increasing input queue waiting time. Many sites employ a combination of several criteria. Depending on the configuration of the site scheduler, other users such as non-members of MediGRID can experience greatly different waiting times as our diagrams show.

At site 1, the test jobs almost never had to wait before execution. At the same time, there is no dependency visible between waiting time and the

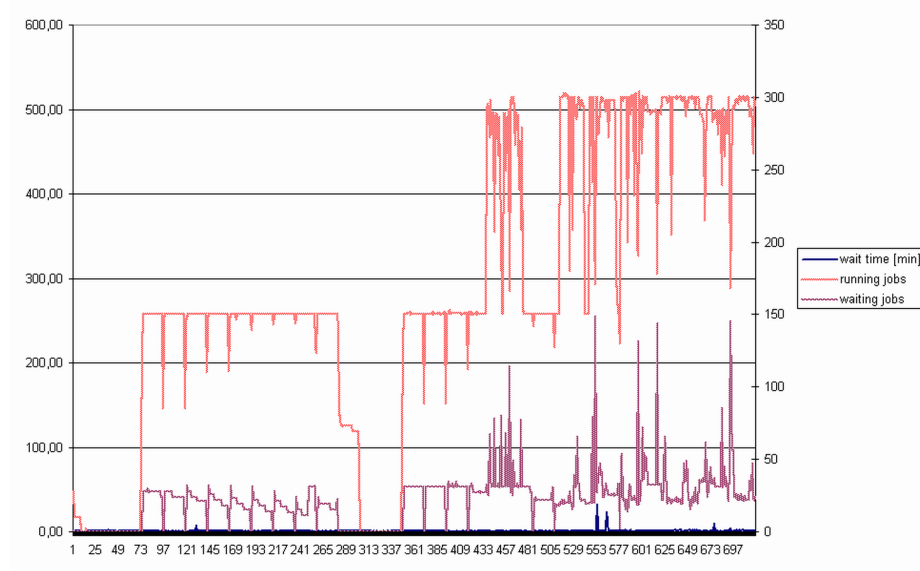


Figure 1: Variation of input queue waiting time and number of jobs in 720 hs at site 1.

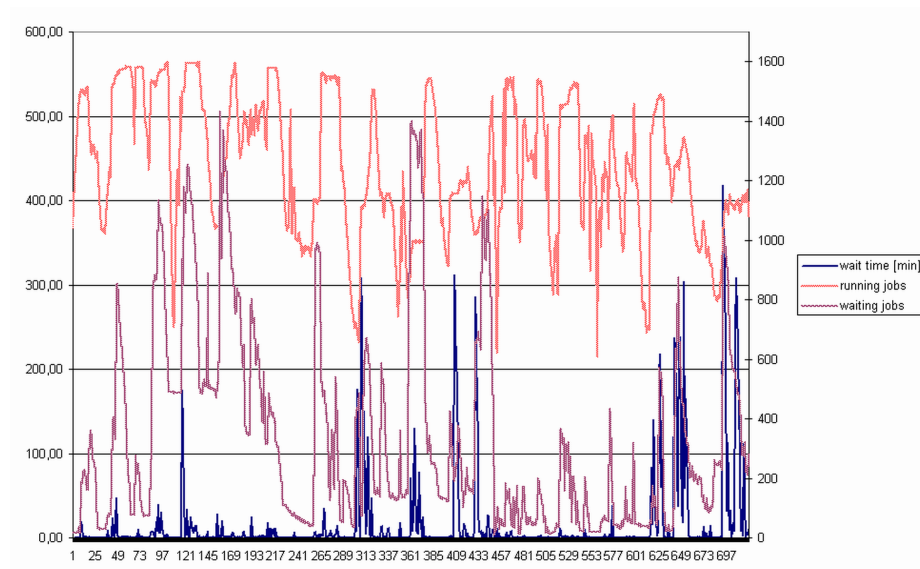


Figure 2: Variation of input queue waiting time and number of jobs in 720 hs at site 2.

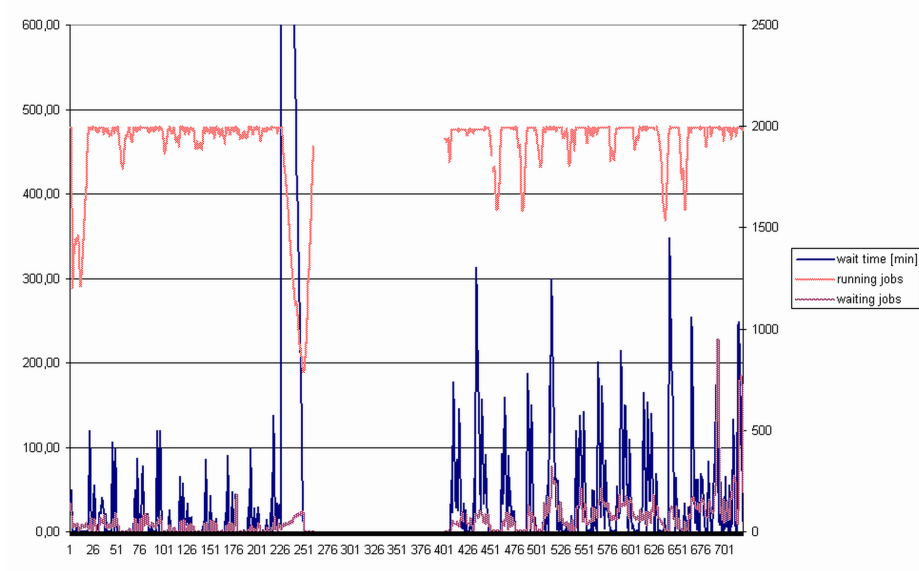


Figure 3: Variation of input queue waiting time and number of jobs in 720 hs at site 3.

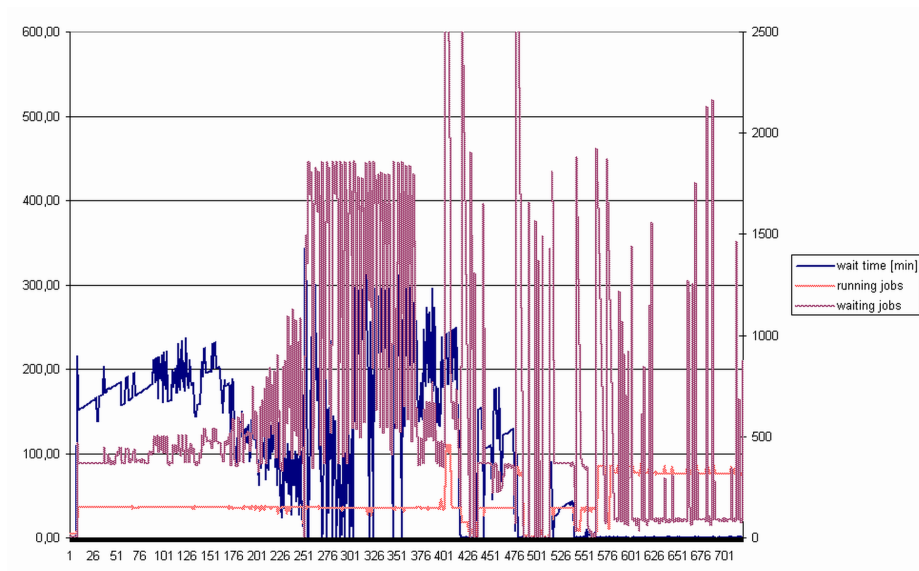


Figure 4: Variation of input queue waiting time and number of jobs in 720 hs at site 4.

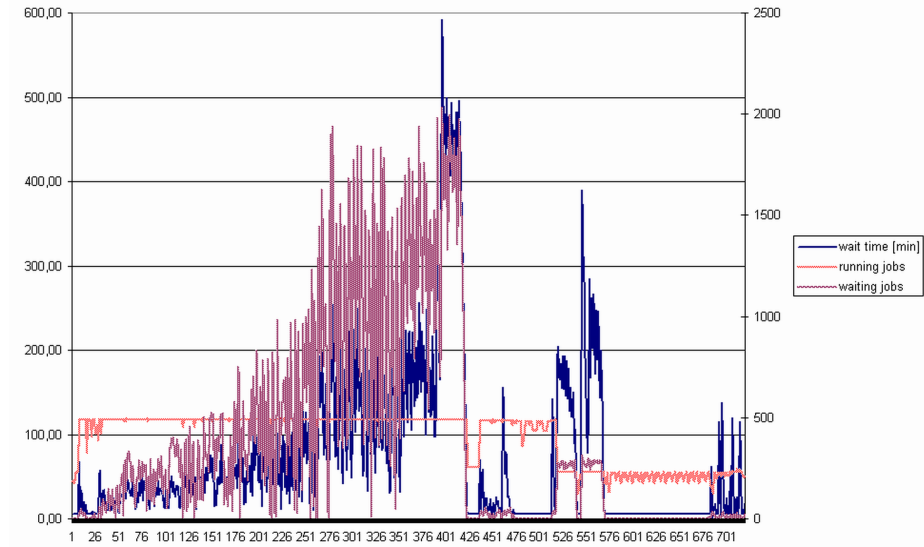


Figure 5: Variation of input queue waiting time and number of jobs in 720 hs at site 5.

number of waiting jobs, i.e. the test jobs were executed instantly even though other many jobs were already waiting in the queue. The maximum number of running jobs increased during the interval. This means that either the number of processor cores was increased or fewer cores were used by jobs submitted later. Because of the missing correlation between waiting time and waiting jobs, the local scheduling of site 1 probably uses the fairshare method or it assigns very high static priorities to MediGRID jobs.

The graphs of site 2 show immense variations in queue waiting times. Long periods with very low waiting times are interrupted by steep peaks which last for a few hours. A dependency between waiting times and waiting jobs only seems to exist during the last peaks in the observed interval. This might be an indicator that these jobs received a similar prioritization as the test jobs. The number of running jobs varies a lot while jobs are still queued which points to changes in the parallelism of jobs. The scheduling of site 2 is certainly fairshare-based.

At site 3, the execution of the measurement program was interrupted for a few days, therefore there is a period without data in the center of the 30 day interval. The waiting times show a very clear periodicity of about 24 hours. Additionally, a clear correlation can be observed between waiting time and waiting jobs. The number of running jobs is almost constant, i.e. jobs are very homogeneous and probably serial jobs. The correlation between wait-

ing time and waiting jobs leads to the assumption that site 3 prioritizes jobs based on the input queue waiting time, i.e. with the first-come first-serve (FCFS) queueing discipline.

During the first seven days of the presented interval, waiting times at site 4 lasted on average three hours and correlated with the number of waiting jobs. Afterwards both values show high fluctuation and no consistent correlation. In the last seven days, the test jobs experienced almost no waiting time even though other jobs were waiting in the input queue. The number of running jobs is very constant with a step at the beginning of the last week. The scheduling at site 4 appears to use mainly FCFS, except from the last week. Then, it is changed to fairshare or high prioritization for MediGRID jobs.

The graphs of site 5 show an increasing waiting time, together with an increase in the number of waiting jobs. At the peak, waiting time is almost ten hours, then both values settle down to approximately zero. The second half of the interval is characterized by peaks in waiting time that still correlate with the number of waiting jobs. The number of running jobs remains constant during most of the interval. The scheduling of site 5 is certainly dominated by the FCFS principle.

4 Current Scheduling in MediGRID

In MediGRID, the execution of complex workflow applications consisting of several program executions is handled by the Grid Workflow Execution Service (GWES) [11]. This workflow management system is used in many European Grid projects, e.g. MediGRID, Instant-Grid, BauVOGrid, K-Wf Grid and CoreGRID [19], [4], [3], [14], [6].

The workflow model for the GWES environment is based on Petri nets. Petri nets consist of places (symbolized by circles) and transitions (symbolized by squares). Places and transitions are connected by directed edges. They always alternate with each other. In the workflow, transitions stand for conditional activities while places represent input and output data. Places contain so-called tokens as soon as the input data are available. Transitions can only be activated if all input places of the transition have a token. When the transition is started, one token from every input place of the transition is taken, and when the transition is completed one token is given to every output place. In this way, Petri nets model control and data flow between application components. MediGRID workflows are specified with the XML-based Grid Workflow Description Language (GWorkflowDL).

The workflow management infrastructure consists of several components depicted in Fig. 6. The central service is the GWES which processes the workflow documents. The first action after initialization is the resource matching. GWES includes a Resource Matcher that searches for available instances

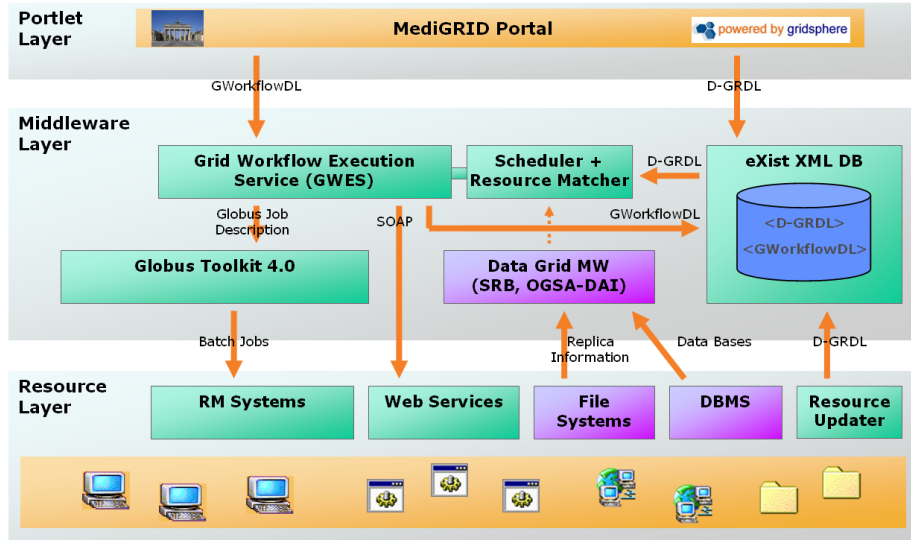


Figure 6: Architecture of the MediGRID middleware.

of the program classes specified in the workflow. For this, GWES communicates with an eXist resource database that contains information about software instances available on the machines in the Grid. The resource database also contains machine descriptions with information about the utilization of the machines. Both types of information are specified by means of the XML-based D-Grid Resource Description Language (D-GRDL). Additionally, the eXist database stores the GWorkflowDL documents of active and completed workflows.

The scheduling in MediGRID is accomplished by the scheduler integrated in GWES. When a transition is ready to execute, i.e. when all input data are available, GWES automatically selects one suitable machine, based on current utilization. Machine utilizations are periodically collected by a Resource Updater and stored in the resource database. Then, the scheduling algorithm calculates a quality value, i.e. metric, between 0 (busy) and 1 (idle) for all resources. The calculation differs between workstation and cluster resources. Workstations are single machines that are run in time sharing. In this case, the scheduler uses the "one minute load" for the calculation [12]. For clusters instead, which are used in space sharing, the metric is calculated by means of the number of running and waiting jobs. In the end, the scheduler chooses randomly one resource from the sublist of resources whose metric is greater than a given threshold value (e.g. 0.6). The selection is randomized to prevent an overload of machines in case of outdated or insufficient monitoring data. This way, the scheduler provides a rough load-balancing

between resources and tries to improve the job throughput of the Grid.

After the scheduling, GWES performs an automatic file stage-in, i.e. it organizes all necessary data transfers for the chosen resource. For file transfers, GWES uses the RFT [1] service from the underlying GT4 middleware. Then, GWES either invokes a web service on the selected resource or it creates a job description and submits a job there which executes a command-line program. The job submission is done via WS-GRAM [2], the job manager of GT4. If a file transfer fails or the job has not finished successfully, GWES performs a fault management and reschedules it onto another machine.

4.1 Criticism of Current Scheduling

The current scheduler implementation in MediGRID belongs to the class of just-in-time algorithms, which make the planning only on locally reasonable decisions. Research shows [23] that just-in-time algorithms can produce good results for independent Grid jobs and rather simple workflows, provided that accurate resource performance information is available. However, they do not provide any full Petri-net analysis for task dependencies, i.e. they do not consider the order of task execution for scheduling. Therefore, just-in-time schedules lack performance in complex application workflows that have many concurrent tasks. This affects especially the important class of unbalanced (asymmetric) workflows with parallel threads that differ significantly in expected thread execution times. In this case, preference has to be given to the longer threads to allow all threads to finish within similar time. The execution of such workflows can be improved by employing a full-graph scheduling as performed by the HEFT algorithm.

5 Tier 1 of the New Approach: Workflow-level Scheduling

5.1 HEFT

Our new approach is based in its first tier on HEFT [21]. HEFT is an extension of the classical list scheduling algorithm based on directed acyclic graphs (DAG) for heterogeneous environments. HEFT is a simple and computationally inexpensive algorithm, which schedules a workflow by “backward” traversing the directed graph from output to input, constructing an ordered list of tasks, and mapping the tasks to resources.

The HEFT algorithm consists of 3 phases [23]:

1. Weighting: it assigns weights to the nodes and edges in the graph;
2. Ranking: it creates a sorted list of tasks, ordered by their priorities;

3. Mapping: it assigns tasks to resources.

In phase 1, the weights assigned to nodes correspond to the predicted execution times of the tasks, while the edge weights correspond to the predicted data transfer times between the resources. HEFT assumes these times to be known. In environments with homogeneous resources, the weights directly reflect the predicted times. In heterogeneous environments, the weights must be adjusted considering variances in execution times on different resources, and different data transfer times on data links. Several adjustment methods were proposed and compared. Each of them provides another accuracy with respect to the considered scenario. The common method is to take the arithmetic average over all resources.

In the ranking phase 2, the workflow graph is traversed backward, and a rank value is assigned to each of the tasks. The rank value denotes the task's priority, thus a higher rank means a greater priority. The rank of a task is equal to the tasks' weight plus the maximum successive weight. This means for every edge leaving the task, that the edge weight is added to the previously calculated rank of the adjacent node, and that the maximum of the summations is chosen. In the end, the tasks are sorted by decreasing rank order. This results in an ordered ranking list.

In the mapping phase, tasks from the ranking list are mapped to the resources one after the other, and each task is assigned to that resource which minimizes the task's earliest expected finish time.

Fig. 7 shows the ranks of HEFT for a simple example workflow graph. a and b are weights and c is the calculated rank. a denotes the average predicted execution time, and b denotes the average predicted data transfer time. In the example, the weights are arbitrary. The ranking of tasks is: B, A, E, D, C, F, G and H.

5.2 Our Modifications of HEFT

For the application on Petri net graphs, some modifications of the HEFT algorithm are necessary: a) Transitions are not connected directly with each other, but separated by places. b) Grid data transfer is performed only when a token moves from the place to the next transition. That is why the weight of the transfer is written atop the input edges of the transition. c) Output edges do not have weight values.

In contrast to DAGs, Petri nets can contain control flow transitions which contain conditions for firing and loops. Both can only be evaluated at runtime. When the full-ahead workflow-scheduling is applied, control flow constructs are ignored, because evaluation cannot be made beforehand. This way, the Petri net branch with the higher weight will be considered for the ranking. If the branch leads to a loop, the loop has to be detected to prevent the traversal of the graph from looping forever. For the ranking, we assume

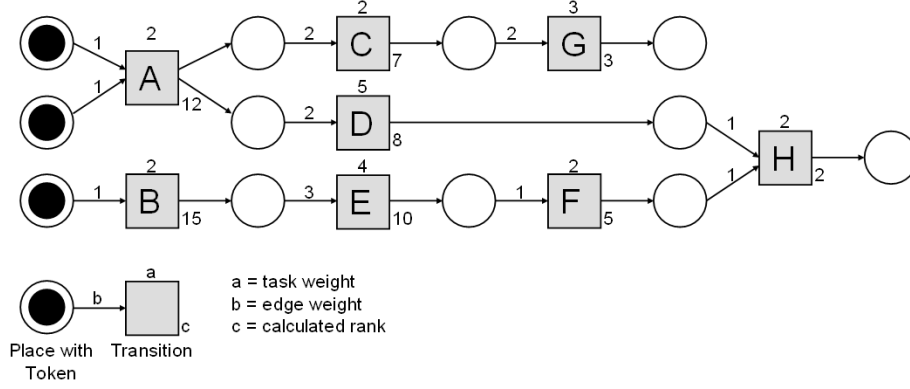


Figure 7: Example workflow graph with weights and ranks calculated by HEFT.

loops to be taken exactly once. This is achieved by virtually unrolling for one iteration of the loop.

The value of the predicted execution time a of a transition is derived from previous executions of the same task. To this end, the runtime of each job is measured during execution. As the job runtime can vary very much, we use the method of exponential smoothing to estimate the next value based on previous values. Exponential smoothing is equivalent to a first-order infinite impulse-response low-pass filter. The prediction value is calculated as $a_t = \alpha * m' + (1 - \alpha) * a_{t-1}$, where $m' = m * p$. m denotes the latest measured execution time, and a_{t-1} is the previous predicted value of a . p is the performance factor of the resource and used to normalize m with respect to a reference system with 2.66 GHz Intel Xeon processors. p has to be calculated by means of SPEC benchmarks for all Grid resources. $\alpha \in [0, 1]$ is called smoothing factor. We use $\alpha = 0.3$ which gives more weight to the previously predicted values for a stronger smoothing. The method of exponential smoothing also facilitates the integration into the existing GWES environment because only one value of m together with the current value of a have to be stored in the D-GRDL description of the software-instance in the resource database.

5.3 Data Transfer

The data transfer time depends on the data size and the bandwidth of the network link connecting the resources. Predicting these values ahead of the workflow execution is difficult because neither data size nor the sites between which the data transfer will take place are known. Additionally, data transfer rates can vary during workflow execution time. Because of this, predic-

tion of b would require a large set of measurements covering multiple previous values for all combinations of sites. This would imply major changes in the already existing resource database. Therefore, we assume instead for the workflow-level scheduling that tasks produce output data in the order of 10 MB, and that bandwidth behaves as a normally-distributed random-variable with a mean of 10 Mbit/s and a variance 4. The transfer setup time is assumed to be 2 seconds. These assumptions are supported by practical experience and result in an average transfer time of 10 seconds.

5.4 Queue Waiting Time

Another aspect that has to be considered are the potential queue waiting times that are imposed on tasks at resources and that lead to a prolongation of pre-execution time. Many Grid-scheduling research-groups assume a Grid model with high availability and good control over the resources by the scheduler, which is often the case for scientific workflows executed in research institutions. However, as our measurements in section 3 have shown, such assumptions do not hold for the D-Grid environment where availability of resources is limited and where no control exists over the site-level resource-management systems. This is a fundamental constraint that limits the possibilities of meta-scheduling in D-Grid.

These circumstances lead to the conclusion that static full-ahead workflow scheduling alone is inappropriate in the D-Grid environment. Therefore, we only employ the first two phases of the HEFT algorithm to calculate static priorities for the Petri net transitions before the actual processing of the Petri net starts. For the mapping phase that assigns tasks to resources, we employ a greatly improved version of the existing dynamic just-in-time scheduling of MediGRID.

6 Tier 2 of the New Approach: Grid-level Scheduling

6.1 Task Prioritization

The second tier in the proposed approach performs a multi-criteria just-in-time Grid scheduling based on dynamic resource data, short-time predictions and additional information available at runtime. During the workflow processing, transitions that are ready to execute are placed in a queue within GWES. While GWES processes many workflows at the same time, this internal queue holds tasks from several workflows belonging to different users. Initially, the tasks are ordered by the time at which they became executable, i.e. when all preceding transitions are finished.

Tier 2 of the New Approach: Grid-level Scheduling

While the only optimization goal of the workflow-level scheduling is to decrease the execution time of a single workflow, joint Grid-level scheduling of all tasks allows for grid-wide, i.e. for global improvements. In every scheduling cycle, all tasks in the internal queue are rearranged depending on the prioritization policies and are mapped to the available resources. In the new approach, the prioritization is split up into four hierarchical levels: urgency, user, workflow and transition level.

On the transition level, the default operation principle is to sort tasks according to their ranks calculated in phase 1 and 2 of the HEFT algorithm. This provides an improved performance for unbalanced workflows. Alternatively, tasks can be sorted by static priorities that are defined in the workflow description.

After the first sorting, all tasks are sorted a second time with regard to the workflows they originate from. Workflows are sorted chronologically by default. First entries are all tasks from the earliest started workflow, followed by the tasks from the second eldest workflow and so on. Among the same workflow, tasks keep their positions of the first sorting. This method meets the user's typical expectancy that when he starts many workflows one after the other, the latter workflows should not unnecessarily delay the first ones. That would happen if tasks were not sorted by workflows because the HEFT algorithm assigns the highest rank to the first tasks within a workflow. Alternatively, workflows can be sorted also by static priorities that are defined in the workflow description.

The third stage of our method sorts the tasks with respect to the users who submitted the workflows. To improve fairness between MediGRID users, a fairshare technique is employed additionally that continuously accounts resource usage within a fairshare window of 14 days. Then, tasks from different users are sorted in ascending order with the users' resource utilization. As an alternative, static priorities can be defined for MediGRID users. The order of workflows and ranks from the previous stages is again preserved. The final stage of our method treats tasks from time-critical workflows and places them at the beginning of the internal queue. Such tasks can e.g. belong to interactive workflows that are marked with a special "urgent" flag.

All stages can also be disabled on request which means no sorting will be done for transition, workflow, user or urgency criteria. For example, the current implementation equals a sorting by static transition priorities only. After the prioritization, the tasks are submitted in their final order to resources best currently available. If sufficient resources are available then no tasks are artificially detained in the internal queue. Our most important goal in resource selection is the reduction of execution time that is extended by Grid-wide optimizations. Similar to the current implementation, the scheduler will try to balance load across the resources and improve thereby job throughput.

6.2 Queue Waiting Time Predictions

As described in section 4, the current MediGRID scheduling randomly selects a resource whose quality value is greater than the threshold. We propose to improve resource selection with regard to data locality. Data transfer times can be avoided if tasks are scheduled onto compute resources that already have the required data available. This requires the introduction of an interface between GWES and the Data Grid middleware. If a data transfer is performed, preference will be given to that resources with a fast network connection to the data source. To this end, the data transfer rates between the MediGRID sites are continuously measured using the Grid benchmarking service Jawari [7]. During just-in-time scheduling, source and destination of the transfer as well as data size are known, therefore it is possible to predict transfer time as $t_{transfer} = t_{setup} + size/bandwidth_{(source,dest)}$.

Our test measurements in D-Grid have shown that variance in bandwidth is acceptable for short-time predictions. However, we found in contrast to common opinion that the bandwidth is usually not equal in both transfer directions between D-Grid sites. After data transfer, the replica has to be registered in the Data Grid middleware so that it is available for future job executions.

Due to differences in queue waiting times, it will often be advantageous to transfer data and to perform the computation on a resource with less queueing delay. However, to correlate transfer and queueing delay, the current algorithm that calculates a quality metric in $[0,1]$ has to be replaced by a new method that allows to estimate queue waiting time. We propose this method that combines three estimations s , u and v that are based on three different concepts to predict queue waiting time. For all resources, the performance of the three estimations is evaluated and that estimation is used which performed best in the latest scheduling cycles, i.e. the predicted value d_{queue} will be either s , or u or v depending on the accuracy of the previous estimations. All three estimated values are stored for each resource in the D-GRDL machine description in the resource database.

The first estimation, s , works similar to the prediction of job execution time described in the previous section. It uses exponential smoothing to estimate queue waiting time based on previous measurements. The calculation is $s_t = \beta * n + (1 - \beta) * s_{t-1}$, where n denotes the latest measured queue waiting time, and s_{t-1} is the previous estimated value of s . β is the smoothing factor. It is set to $\beta = 0.3$ which has shown by practical experience to sufficiently compensate fluctuations. Multiple values of s should be calculated for different job durations because some sites have exclusive nodes for short and long running jobs or employ backfilling.

The second estimation, u , uses Little's law [10] from queueing theory to calculate the expected value measure of queue waiting time. Application of Little's law requires to determine two basic parameters of the queueing sys-

tem, the average rate of jobs entering the queueing system λ , and the average rate of serving jobs μ . As mentioned before, the Grid scheduler does not have insight into the LRMS and current LRMS do not provide these values. Therefore, we have developed a procedure to measure λ and μ that requires only user permissions.

As described in section 3, a measurement program was installed on all MediGRID clusters which periodically retrieves the number of running and waiting jobs and the identifier of the latest job. The LRMS uses consecutive numerical values as job identifiers. To reliably determine the identifier of the latest job, the program submits a test job and reads its own job number. The difference between the values of the current and the previous measurements gives the arrival rate λ of jobs in the cluster during a time interval. The total service rate for all servers μ' can be calculated by adding the number of jobs that have arrived during the time interval to the total number of jobs having been previously in the system, and by subtracting the total number of jobs currently in the system.

$$\begin{aligned}\lambda &= job_number_t - job_number_{t-1} \\ \mu' &= running_{t-1} + waiting_{t-1} + \lambda - (running_t + waiting_t)\end{aligned}$$

Now we can apply Little's law which relates the steady-state mean system-sizes to the steady-state average waiting-times. A steady-state denotes the situation of the system after it has been settled, i.e. after running for a long time. Prerequisite for reaching a steady-state is that $\rho = \lambda/c\mu = \lambda/\mu'$ (often called traffic intensity) must be strictly less than 1. c is the number of parallel servers. When $\rho > 1$ the queue size never settles down and there is no steady state possible.

According to [10] we define N as the total number of jobs in the system, N_q as the number of jobs waiting in queue, T as the total time a jobs spends in the system, and T_q as the time a job spends waiting in the queue. Then we define the expected value measures of these terms. $L = E[N]$ is the mean number of jobs in the system, $L_q = E[N_q]$ is the mean number in queue, $W = E[T]$ is the mean waiting time in the system, and $W_q = E[T_q]$ is the mean waiting time in queue. Little's laws for general queueing systems (G/G/c) are $L = \lambda W$ and $L_q = \lambda W_q$. Thus, we get the expected queue waiting time as $u = W_q = L_q/\lambda$.

The third estimation, v , is based on a Fourier analysis [16], [5] of the series of measured input-queue waiting-times. We perform a Discrete Fourier transform (DFT) that decomposes the sequence of time values into components of different frequencies. In the frequency domain, the sinusoidal basis functions of the decomposition are analyzed with respect to their amplitudes. Only significant amplitudes are preserved. Amplitudes of low magnitude are set to zero. Afterwards, the frequency domain is transformed back

into time domain using the inverse DFT. Since the DFT only generates frequency components needed to reconstruct the finite time segment that was analyzed, its inverse transform cannot reproduce the entire time domain, unless the input happens to be periodic. For this estimation, we assume that the input data is periodic. Thus we obtain the estimated value v as the first value of the transformed periodic extension of the output sequence. The alteration of amplitudes in frequency domain is made to emphasize the periodicity of the sequence.

6.3 Evaluation of Waiting Time Predictions

To judge the three estimation approaches, calculations were performed with queue data acquired by hourly measurements in the D-Grid. For Little's formula, it turned out to be advantageous to use the current queue length for L_q , and to average λ over the last 12 hours. Figs. 8-12 show exponential smoothing and queueing theory-based estimations in comparison to the actual measured queue waiting times for all five resources during an interval of 288 hours (12 days). The Fourier analysis was done using Excel [8], [15]. An example for DFT data is shown in Figs. 13, 14.

Estimation 1 performs very well for site 1 (see Fig. 8) because the queue waiting times are nearly constant. However, the peaks are predicted one step later than the original which is inherent to first-order filtering. Additionally, after the peaks the prediction fades only slowly. The latter could be improved by choosing a greater smoothing factor, but this would imply more jitter which is not desirable. Estimation 2 performs very bad and is not applicable to site 1.

For site 2 (see Fig. 9), estimation 1 performs reasonable, but suffers even more from the problems described before. Estimation 2 is not able to deliver a useful prediction for site 2.

At site 3 (see Fig. 10), estimation 1 still seems alright at first sight, but the deviation increases with the number of peaks. Estimation 2 performs much better with site 3. Even though it is unable to predict the correct height of the peaks, it does not have a delay as estimation 1. This is especially noticeable at the end of the peaks. The incorrect prediction of the peaks' exact height is a minor issue because as long as the presence of a peak is detected, no jobs are sent to the site anymore.

For site 4 (see Fig. 11), estimation 1 performs reasonable. This site illustrates the importance of the smoothing factor for the suppression of jitter in the queue waiting times. Estimation 2 performs very well in the beginning. In the second half, the magnitude of waiting time is predicted lower than the actual values. However, whenever waiting time decreases estimation 2 signalizes this earlier than estimation 1.

At site 5 (see Fig. 12), estimation 1 performs similar as with site 4. The

Tier 2 of the New Approach: Grid-level Scheduling

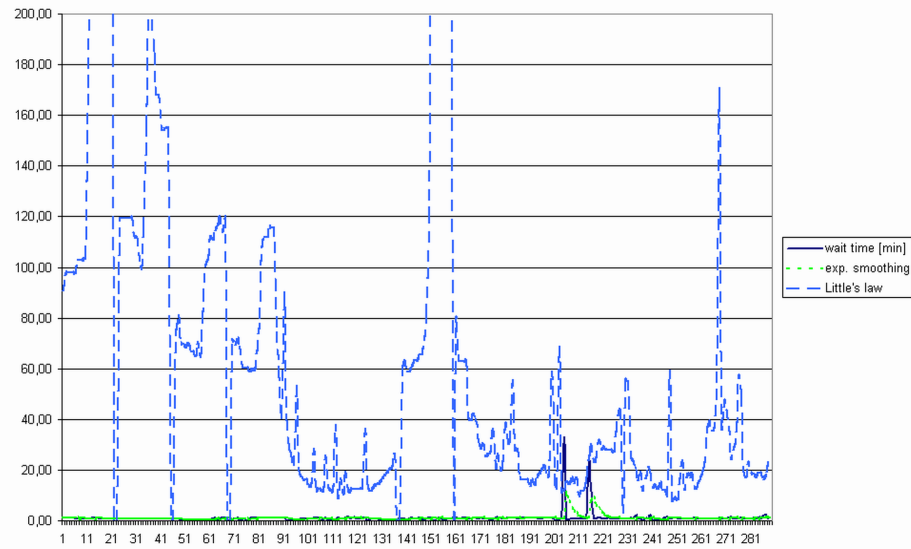


Figure 8: Comparison of measured input-queue waiting-time and estimations for site 1.

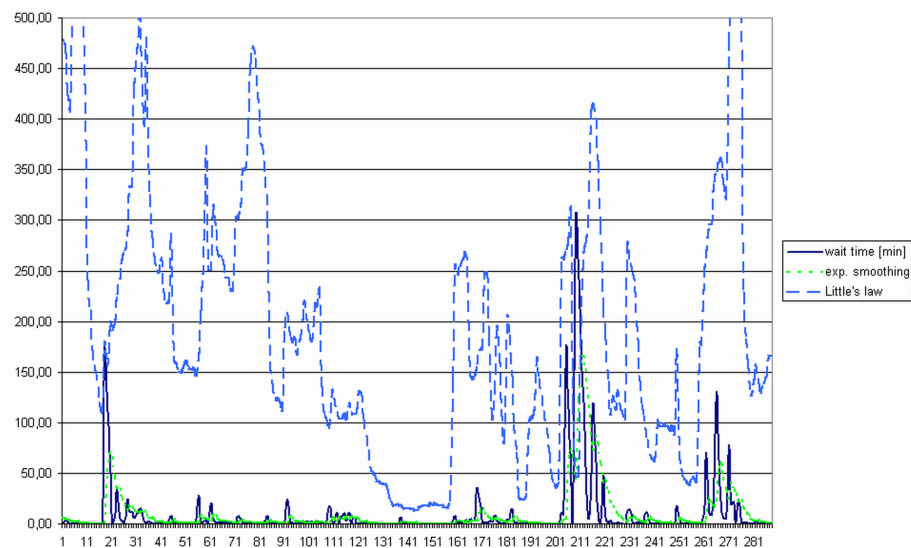


Figure 9: Comparison of measured input-queue waiting-time and estimations for site 2.

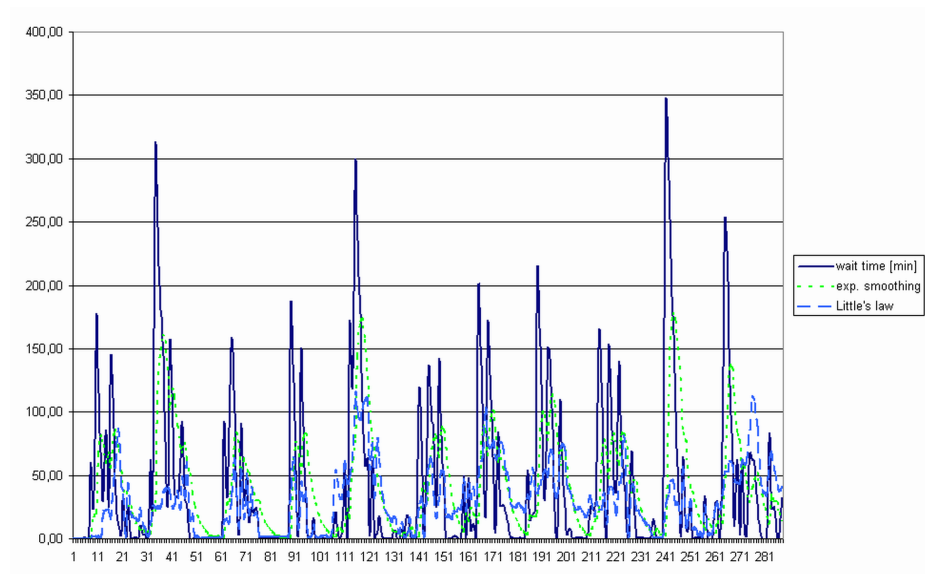


Figure 10: Comparison of measured input-queue waiting-time and estimations for site 3.

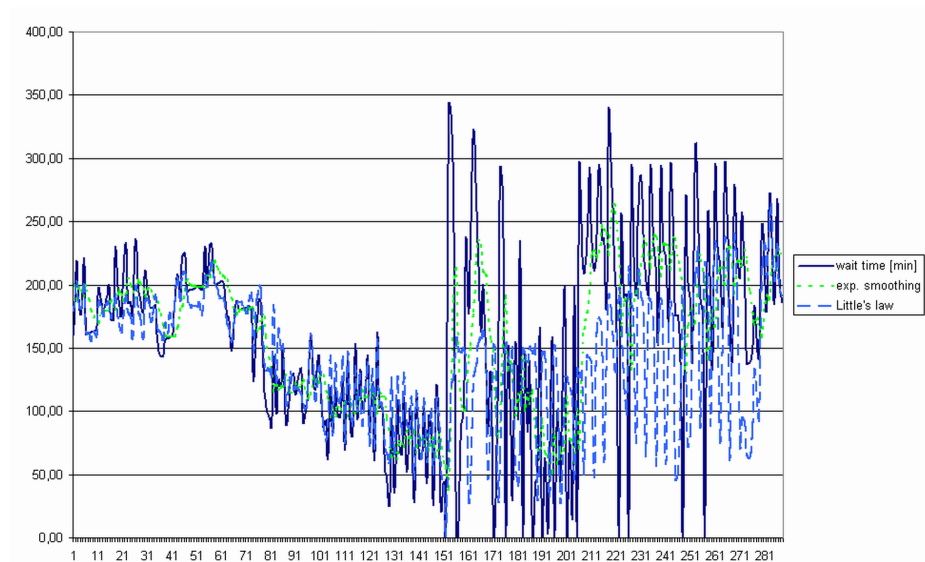


Figure 11: Comparison of measured input-queue waiting-time and estimations for site 4.

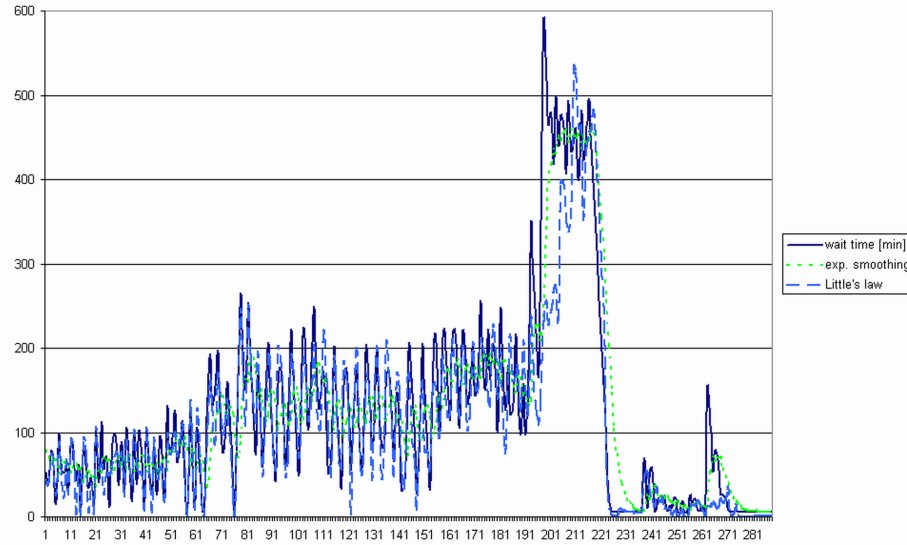


Figure 12: Comparison of measured input-queue waiting-time and estimations for site 5.

main issue is the big delay in descent after the dominant peak in the diagram. Estimation 2 performs very well, only the very last peak is not really predicted.

In Fig. 13, the Fourier spectrum of waiting times of site 3 is shown with original amplitudes as well as with clipped values. The periodicity of the time series data is clearly noticeable from the dominant peak in the frequency spectrum that appears at $f > 0$. The peak occurs at the tenth sinusoidal basis-function frequency because we have sampled ten periods in the original time sequence. All other frequencies (except $f = 0$ which denotes the irrelevant DC part of the signal) have much smaller amplitudes in the spectrum. After having obtained the spectrum, we applied a clipping procedure as follows to the amplitudes: If the magnitude is less than 10% of the maximum amplitude the respective amplitude is set to zero. The periodic extension shown in Fig. 14 resembles the original continuation very well. It predicts the following value better than estimation 1 and 2. It should be pointed out again that only the first yellow point is used from one DFT calculation. The following values are calculated by shifting the input window one step right and repeating the calculation. As can be seen in the graph, the inverse transform can contain negative values, i.e. queue waiting times. Those are also clipped to zero.

The depicted measurements and the numerical prognosis showed that all

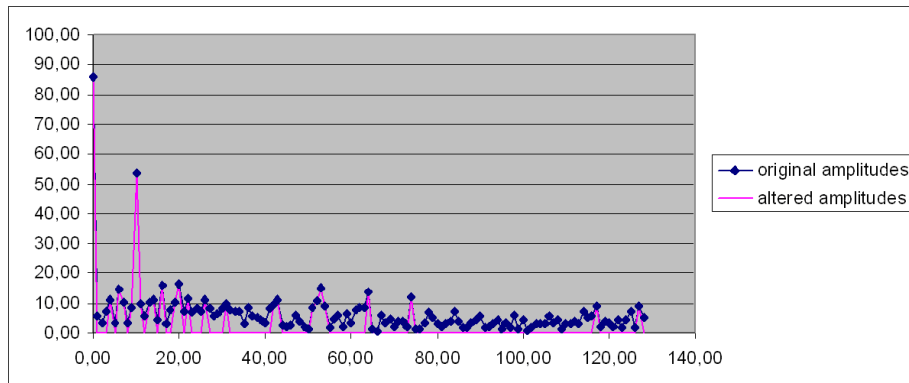


Figure 13: Frequency spectrum for a sequence of waiting times from site 3.

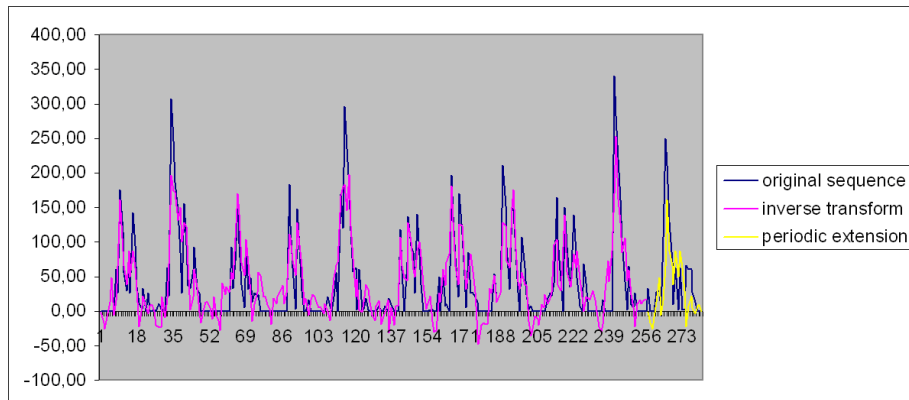


Figure 14: Original sequence of waiting times from site 3 and the inverse transform of altered amplitudes.

three approaches (Exp. smoothing, Little and Fourier analysis) to estimate queue waiting times, cover each one scenario best. Queueing theory is especially capable to detect changes quickly and to predict low values correctly. It performs the better, the more the LRMS configuration matches the first-come first-serve queueing discipline. It becomes unreliable when the arrival rate is very low, or when the actual waiting times are high (more than two hours). Exponential smoothing instead reproduces the trend and the exact value of waiting times better than queue modeling. It is the best choice for resources that employ mainly fairshare scheduling or static priorities. However, it suffers from the delay inherent to first-order filtering and needs frequent measurements of the actual queue waiting times. Finally, Fourier analysis avoids the delay of exponential smoothing and is appropriate best for sites that show periodic behavior in queue waiting times.

6.4 Judging Middleware Overhead

Another factor that adds to pre-execution delay is the middleware overhead caused by Globus Toolkit 4 e.g. It is introduced by middleware components such as the execution management and security infrastructure services involved in job submission. However, this overhead time is almost identical on all sites because all use the same middleware. Furthermore, overhead time is much smaller than queueing time because queues are normally full in D-Grid. Therefore, middleware delay can be neglected.

6.5 Summary Discussion

Contemporary workflow scheduling strategies that employ static full-ahead scheduling of the complete workflow graph use sophisticated task scheduling heuristics such as list scheduling or genetic algorithms but have problems to cope with the dynamic behavior of the Grid. Therefore, they require frequent rescheduling during the execution of the workflow. An important quantity here is the expected delay until jobs can start to execute. This quantity is the sum of the data transfer time $t_{transfer}$ and the queueing delay d_{queue} . Our proposed scheduling method assigns tasks to resources based on this expected delay. This procedure is similar to the mapping phase of the HEFT algorithm.

However, the repetition of the workflow-level scheduling due to phase 1 and 2 of the HEFT algorithm is beneficial for example in case of a system failure during job execution before the fault management of GWES resubmits the task to another machine to repeat execution. Another example for beneficial repetition of the workflow-level scheduling due to phase 1 and 2 of the HEFT is when a loop in the Petri net graph is entered to correct the ranking of the tasks.

It has to be noted that for applying the HEFT algorithm to Petri nets, some modifications of the algorithm are necessary to handle non-DAG characteristics of Petri nets such as control flow transitions and loops. With the just-in-time Grid scheduling which is tier two of our new approach, GWES keeps the capability to handle all Petri-net control-flow constructs as shown in the current implementation.

Finally, the proposed two-tier approach is downward compatible, and it improves the scheduling performance in case of complex, unbalanced application workflows. Even if the predictions of execution times are incorrect, the performance is expected to be similar to but never worse as the current scheduling method.

Table 1 compares the existing MediGRID algorithm with the proposed novel approach using the taxonomies from [22].

	current implementation	Our novel approach
multiple criteria	(x)	x
multiple workflows	x	x
full-ahead planning	-	x
just-in-time	x	x
workflow optimization	-	x
grid-wide optimization	x	x

Table 1: Comparison of scheduling methods.

7 Conclusion

As a result of our measurements we have made in D-Grid, we have disclosed four intrinsic problems for meta-scheduling that should be considered in future designs of Grids, otherwise Grid-level scheduling will always remain insufficient. These problems are: competing-schedulers, lack-of-control, information-insufficiency, and the non-continuously differentiable-function problem. This means that Grid scheduling in D-Grid will ever be a difficult task, because Grid schedulers only have the role of “power users” competing with other users. They have no direct control over the LRMS, and the information available to base the scheduling decisions upon is very limited. Finally, many resources are highly utilized so that queue waiting times can last up to hours.

Since our task was to design a meta-scheduler under these boundary conditions we found out that three different site scenarios could be identified for which three disjoint estimation methods could be given that predict the respective situation best. By this, scheduling performance can be significantly improved.

Our methodology for the scheduling in MediGRID combines historical information that was gained in previous job executions with information from periodic online measurements. This approach replaces the current just-in-time Grid scheduling of GWES by an algorithm that makes scheduling decisions based on estimations of execution delay. Additionally, it introduces a preceding workflow-level scheduling-phase that employs parts of HEFT, as well as performance predictions by full-graph analysis.

Future work on the two-tier approach is the practical implementation and test of the proposed scheduling method in MediGRID to show that the new methods work also in a real production environment.

8 Acknowledgment

This work is partly funded by the German Federal Ministry of Education and Research (BMBF) within the MediGRID project, grant numbers 01AK803A-H.

References

- [1] The Globus Alliance. Reliable File Transfer Service (RFT). <http://www.globus.org/toolkit/docs/4.0/data/rft/>.
- [2] The Globus Alliance. Web Service Grid Resource Allocation and Management (WS GRAM). <http://www.globus.org/toolkit/docs/4.0/execution/wsgram/>.
- [3] BauVOGrid. <http://www.bauvogrid.de>.
- [4] C. Boehme, T. Ehlers, J. Engelhardt, A. Felix, O. Haan, T. Kalman, B. Neumair, U. Schwardmann, and D. Sommerfeld. Instant-Grid: Fully Automated Middleware-Deployment Using a Live-CD. In *ICNS '06: Proceedings of the International conference on Networking and Services*, page 70. IEEE Computer Society, 2006.
- [5] T. Butz. *Fourier transformation for pedestrians*. Springer, 2006.
- [6] CoreGRID - Network of Excellence. <http://www.coregrid.net>.
- [7] E. de Oliveira. Jawari - A Grid Benchmarking Service. In *Proceedings of the German e-Science Conference*, Baden-Baden, Germany, 2–4 May 2007.
- [8] P. T. Debevec. The Excel FFT Function v1.2., 2008. http://online.physics.uiuc.edu/courses/phys401/spring09/Lectures/Excel/The_Excel_FFT_Function_v1.2.pdf.

- [9] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. *Lecture Notes in Computer Science*, 3165:11–20, 2004.
- [10] D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris. *Fundamentals of queueing theory*. John Wiley & Sons, 2008.
- [11] A. Hoheisel. Grid Workflow Execution Service - Dynamic and interactive execution and visualization of distributed workflows. In *Proceedings of the Cracow Grid Workshop*, volume 2, pages 13–24, 2006.
- [12] A. Hoheisel and H. Rose. Konzept für das Scheduling von Workflow-Aktivitäten in Instant Grid. Technical report, Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik, June 2005.
- [13] E. Huedo, R.S. Montero, and I.M. Llorente. The GridWay framework for adaptive scheduling and execution on Grids. *SCPE*, 2006:2007, 2004.
- [14] K-Wf Grid. <http://www.kwfgrid.eu>.
- [15] L. Klingenberg. Frequency Domain Using Excel, 2005. <http://online.sfsu.edu/larryk/Common%20Files/Excel.FFT.pdf>.
- [16] N. Morrison. *Introduction to Fourier analysis*. Wiley New York, 1994.
- [17] J. Nabrzyski, J.M. Schopf, and J. Węglarz. *Grid resource management: state of the art and future trends*. Kluwer Academic Publishers, 2003.
- [18] R. Prodan and T. Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694. ACM New York, NY, USA, 2005.
- [19] D. Sommerfeld, T. Lingner, M. Stanke, B. Morgenstern, and H. Richter. AUGUSTUS at MediGRID: Adaption of a bioinformatics application to grid computing for efficient genome analysis. *Future Generation Computer Systems*, 25(3):337 – 345, 2009.
- [20] Condor Team. DAGMan (Directed Acyclic Graph Manager). <http://www.cs.wisc.edu/condor/dagman/>.
- [21] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.
- [22] M. Wiecek, A. Hoheisel, and R. Prodan. *Grid Middleware and Services: Challenges and Solutions*, chapter Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem, pages 237–264. CoreGRID. Springer, June 2008.

References

- [23] M. Wieczorek, R. Prodan, and T. Fahringer. Comparison of workflow scheduling strategies on the Grid. *Lecture Notes In Computer Science*, 3911:792–800, 2006.